REMARKS/ARGUMENTS

This Amendment and the following remarks are intended to fully respond to the Office Action dated July 27, 2005. In that Office Action, claims 1-51 were examined, and all claims were rejected. More specifically, claims 1-24, 34-39, and 47-51 stand rejected under 35 U.S.C. § 101 because the claimed invention is directed to non-statutory subject matter; claims 1-20, 25-30, 33-38, 40-43, and 47-50 stand rejected under 35 U.S.C. § 102(e) as being anticipated by Blais et al. (USPN 6,505,344), hereinafter "Blais"; and claims 21-24, 32, 32, 39, 44-46, and 51 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Blais in view of Pinter et al. (USPN 6,457,023), hereinafter "Pinter". Reconsideration of these rejections, as they might apply to the original and amended claims in view of these remarks, is respectfully requested.

In this Response, claims 1, 25, 34, 35, 40, and 47 have been amended and no claims have been added or canceled.

Claim Rejections - 35 U.S.C.§ 101

Claims 1-24, 34-39, and 47-51 stand rejected under 35 U.S.C. § 101 because the claimed invention is directed to non-statutory subject matter. Independent claims 1, 25, 34, 35, 40, and 47 have been amended to put the claims in a more appropriate format. As such, the 35 U.S.C. § 101 rejections are now moot.

Claim Rejections – 35 U.S.C. § 102

Claims 1-20, 25-30, 33-38, 40-43, and 47-50 stand rejected under 35 U.S.C. § 102(e) as being anticipated by Blais. Applicant respectfully traverses the § 102(e) rejections because either the Examiner has failed to state a prima facie case of anticipation. A prima facie case of anticipation can be met only where the reference teaches each and every aspect of the claimed invention. See MPEP §§ 706.02 & 2136.

The examiner asserts that Blais teaches the invention as claimed, including configuring a target program to allocate thread specific data to a thread specific heap and allocating shared data to a shared heap.

It is respectfully submitted that there is a fundamental difference between the claimed "thread-specific heaps" in Applicant's application and the "inlined" invocation stack frame disclosed in the Blais patent. While the escape analysis considered by Blais (determining "whether the lifetime of an object 'escapes' the method that creates the object" *see*, col. 6, lines 54-56) is different than the escape analysis described in the present application ("[t]hread escape

analysis, in an embodiment of the present invention, computes for each global value (e.g., reference constant or static field) and its (transitive) fields and array elements a conservative estimate of whether the value is accessed by more than one program thread" page 12, lines 15-17), this difference need not be considered to distinguish the present invention. Simply, in Applicant's claimed invention, the alterations to the target program to allocate the thread-specific objects to thread-specific heaps is different than the changes to the Java code to allocate "escape" objects to the invocation stack frame as described in Blais. Applicant's thread-specific objects are placed in thread-specific heaps and are **not placed in the invocation stack frame** whereas the escape objects in Blais are **placed in the invocation stack frame**. There is simply no suggestion of "a thread-specific heap" in Blais.

In Blais' system, if the escape objects are "no escape" or, in some circumstances, marked *arg* escape, the Java code is modified to allocate the objects in the invocation stack frame. Blais states, at column 8, lines 24-37:

> Referring now to FIG. 11, a method 1100 for generating optimized code represents a simplified flow diagram of some of the functions that are typically included in the generation of the optimized code in step 640. An allocation instruction in the object oriented program is first selected (step 1110). If the allocation instruction is marked as no escape (step 1120=YES), code is generated for the allocation instruction that causes an object to be allocated *on the stack* (step 1130). This allocation is preferably *on the invocation stack frame of the method.* If the instruction is either global escape or arg escape (step 1120=NO), code is generated for the allocation instruction that causes an object to be allocated from the heap (step 1140). (emphasis added)

Blais goes on to state, at col. 8, lines 53-57, "the preferred embodiments provide an improvement to the Choi et al. escape analysis by determining whether an object can be *allocated on a method's invocation stack frame* even though it is marked arg escape."

The system in Blais is dedicated to determining escape objects and allocating those objects either to the method's invocation stack frame or to the shared heap. That is, in Blais' system, the no escape or arg escape objects are allocated to the method's invocation stack frame and global escape objects are allocated to the shared heap. At no time does Blais consider, comment, or create a thread-specific heap to which to allocate thread-specific objects.

In contrast, in Applicant's claimed system, the target program is modified to allocate objects into two types of heaps. Thread-specific objects are allocated to a thread specific heap and shared objects are allocated to the shared heap. Then, the thread-specific heap can be

garbage collected without affecting other threads. Applicant's specification, page 8, lines 1-9, states:

> **During runtime, thread-specific data is allocated to thread-specific heaps, and shared data is allocated to one or more shared heaps.** At some appropriate time during execution, a garbage collector module reclaims memory for unneeded program data from one or more of the target program's heaps. Because no thread-specific data within a thread-specific heap is determined to be reachable from outside the associated program thread, the thread-specific heap can be collected without impacting (e.g., suspending or synchronizing with) the execution of other program threads in the target program. In addition, the impact of garbage collection of the shared heap on program threads can be minimized because the program threads can substantially continue their execution so long as they do not access program data in the shared heap.

The most glaring difference between the Applicant's claimed system and Blais is highlighted in the discussion of Fig. 1 in Applicant's specification. Applicant's specification, page 9, line 3 to page 10, line 2, states:

> The target program 100 includes multiple concurrently executing program threads, such as a program thread 102 and a program thread 112. **The program thread 102 is represented by a run-time call stack of code frames 104, 106, and 108.** Each code frame includes data for a method that is called during program thread execution, as represented by calls 120 and 122. Likewise, a second program thread 112 is illustrated with a runtime stack of code frames 114, 116, and 118 (i.e., a code stack). Each code frame includes data for a method called during program thread execution, as represented by calls 124 and 126. Objects (e.g., variables, data fields and class instances) having lifetimes that do not exceed the lifetime of the associated code frame may be allocated within the stack. For example, a method may include a local data structure that is statically defined in the source code as an array of four integers (e.g., the array may be defined in the C or C++ programming languages using an exemplary instruction "int fooArray[4];". At compile time, the compiler can then configure the target program to include the array within the stack.
>
> In contrast, a developer may not wish for memory to be allocated on the stack, such as when the lifetime of the object exceeds the lifetime of the associated code frame. Therefore, **the developer may include program instructions for dynamically allocating data in memory that is separate from the stack. Such dynamic allocation is accomplished in a heap, such as a heap 110.** In a C programming language, dynamic allocation in a heap may be indicated using the keyword "malloc", for example. In C++, dynamic allocation in a heap may be indicated using the keyword "new", such as in "fooClass fooObject = new fooClass;" where fooClass is the object's class and fooObject is the name of the class instance. In the C++ example, the fooObject instance of class fooClass is allocated in the heap.

and continued at page 11, lines 1-19:

> In FIG. 1, **the heap 110 includes three individual component heaps 128, 130, and 132. The heap 132 (shown by a dashed line rectangular box) is a "shared heap".** As shown, both program threads 102 and 112 reference the object 144 within the shared heap 132. For example, both the current code frame 108 in program thread 102 and the current code frame 118 in program thread 112 contain pointers to the object 144 in the shared heap 132. The object 144 also references another object 142, also allocated in the shared heap 132. As such, both objects 144 and 142 are determined to be reachable from both program threads 102 and 112.
>
> In contrast, **heaps 128 and 130 (represented by dotted line rectangular boxes) are "thread-specific heaps".** In thread-specific heap 130, thread-specific objects 138 and 140 are reachable only from program thread 112. In addition, the object 138 also references the thread-specific object 142, which is allocated in the shared heap 132. Likewise, in thread-specific heap 128, thread-specific objects 133, 134, and 136 are reachable only from the program thread 102.
>
> In an embodiment of the present invention, the specific allocation of an object in the shared heap 132 or one of the thread-specific heaps 128 and 130 is governed by code analysis of the target program 100, such as during compilation. Such code analysis can identify objects that are allocated by a given thread and are determined to be unreachable from outside that program thread. An object that is proven to be reachable by only a single program thread is referred to as a "thread-specific object". In contrast, if the object is deemed potentially reachable by more than one program thread, the object is referred to as a "shared object".

Therefore, in Applicant's invention as claimed, there are three separate components: an invocation stack frame, a shared heap, and a thread-specific heap. This differs substantially from Blais. Blais describes only two components – an invocation stack frame and a shared heap. In addition, the garbage collector algorithms, in the present application, run on the thread-specific heaps and do not change the invocation stack frame.

Blais suggests using inline stacks at compile time to "inline" methods onto the invocation stack frame. Col. 9, lines 17-20 ("[A]n inline stack is a list of methods that must be inlined at particular call sites in order for a given allocation instruction *to allocate objects on the stack, rather than from the heap.*") (emphasis added). The inline stacks are not used for the runtime allocation of objects, and therefore have no relation to the thread-specific or shared heaps in Applicant's claimed system.

Independent claims 1, 25, 34, 35, 40, and 47 each recite " thread-specific heaps" as defined precisely in Applicant's specification. Therefore, it is respectfully submitted that Blais

et al. does not disclose or teach Applicant's claimed invention as defined in these independent claims.

Further, all other claims depend from these allowable independent claims. As such, all other claims, i.e., claims 2-24, 26-33, 36-39, 41-46, and 47-51, are also allowable over the cited prior art.

Conclusion

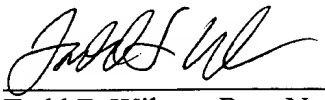
This Amendment fully responds to the Office Action mailed on July 27, 2005. Still, that Office Action may contain arguments and rejections and that are not directly addressed by this Amendment due to the fact that they are rendered moot in light of the preceding arguments in favor of patentability. Hence, failure of this Amendment to directly address an argument raised in the Office Action should not be taken as an indication that the Applicant believes the argument has merit. Furthermore, the claims of the present application may include other elements, not discussed in this Amendment, which are not shown, taught, or otherwise suggested by the art of record. Accordingly, the preceding arguments in favor of patentability are advanced without prejudice to other bases of patentability.

Enclosed is a Petition for a one-month extension of time in this matter. Please charge Deposit Account No. 13-2725 the fee of $120 for this extension. It is believed that no further fees are due with this Response. However, the Commissioner is hereby authorized to charge any deficiencies or credit any overpayment with respect to this patent application to deposit account number 13-2725.

In light of the above remarks and amendments, it is believed that the application is now in condition for allowance and such action is respectfully requested. Should any additional issues need to be resolved, the Examiner is requested to telephone the undersigned to attempt to resolve those issues.

Respectfully submitted,

11/1/05
Date



Tadd F. Wilson, Reg. No. 54,544
Merchant & Gould P.C.
P.O. Box 2903
Minneapolis, MN 55402-0903
(303) 357-1651
(303) 357-1671 (fax)